Kotlin - Visibility Control (Modifiers)

The Kotlin visibility modifiers are the keywords that set the visibility of classes, objects, interface, constructors, functions as well as properties and their setters. Though getters always have the same visibility as their properties, so we can not set their visibility.

> Setters are the functions which are used to set the values of the properties, where as getters are the functions which are used to get the values of those properties.

There are four visibility modifiers in Kotlin:

- public
- private
- protected
- internal

The default visibility is public. These modifiers can be used at multiple places such as class header or method body. Let's look into the detail of these modifiers:

Public Modifier

Public modifier is accessible from anywhere in the project workspace. If no access modifier is specified, then by default it will be in the public scope. In all our previous examples, we have not mentioned any modifier, hence, all of them are in the public scope. Following is an example to understand more on how to declare a public variable or method.

```
class publicExample {
   val i = 1

   fun doSomething() {
   }
}
```

In the above example, we have not mentioned any modifier, so the method and variable defined here, are by default public. Though above example can be written with **public** modifier explicitly as follows:

```
public class publicExample {
   public val i = 1

   public fun doSomething() {
    }
}
```

Private Modifier

The classes, methods, packages and other properties can be declared with a **private** modifier. This modifier has almost the exact opposite meaning of public which means a private member can not be accessed outside of its scope. Once anything is declared as private, then it can be accessible within its immediate scope only. For instance, a private package can be accessible within that specific file. A private class or interface can be accessible only by its data members, etc.

```
private class privateExample {
    private val i = 1

    private val doSomething() {
    }
}
```

In the above example, the class **privateExample** is only accessible from within the same source file and the variable **i** and method **doSomething** can only be accessed from inside of class privateExample.

Example

Let's check a simple example showing the usage of private members:

```
open class A() {
    private val i = 1

    fun doSomething(){
        println("Inside doSomething" )
        println("Value of i is $i" )
    }
}
class B : A() {
    fun printValue(){
        doSomething()
        // println("Value of i is $i" )
    }
}

fun main(args: Array<String>) {
    val a = A()
    val b = B()

    b.printValue()
}
```

When you run the above Kotlin program, it will generate the following output:

```
Inside doSomething
Value of i is 1
```

Here we can not access variable **i** inside class B because it has been defined as private which means it can be accessed inside the class itself and no where else.

Protected Modifier

Protected is another access modifier for Kotlin, which is currently not available for top level declaration like any package cannot be protected. A protected class or interface or properties or function is visible to the class itself and it's subclasses only.

```
package one;

class A() {
   protected val i = 1
}
class B : A() {
   fun getValue() : Int {
      return i
   }
}
```

In the above example, the variable **i** is declared as protected, hence, it is only visible to class itself and it's subclasses.

Example

Let's check a simple example showing the usage of protected members:

```
open class A() {
   protected val i = 1

   protected fun doSomething(){
      println("Inside doSomething" )
      println("Value of i is $i" )
   }
}
class B : A() {
   fun printValue(){
       doSomething()
       println("Value of i is $i" )
   }
}

fun main(args: Array<String>) {
   val a = A()
   val b = B()

   //a.doSomething()
```

```
    b.printValue()
  }
```

When you run the above Kotlin program, it will generate the following output:

```
Inside doSomething
Value of i is 1
Value of i is 1
```

Here we can not call **doSomething()** even using an object of class A because it has been defined as protected which means it can be accessed inside the class itself or in its subclasses only.

Internal Modifier

Internal is a newly added modifier in Kotlin. If anything is marked as internal, then the specific field will marked as the internal field. An Internal package is visible only inside the module under which it is implemented. An internal class interface is visible only by other class present inside the same package or the module. In the following example, we will see how to implement an internal method.

```
package one

internal class InternalExample {
}

class publicExample{
    internal val i = 1

    internal fun doSomething() {
    }
}
```

In the above example, class **InternalExample** is only accessible from inside the same module, similarly variable **i** and function **doSomething()** are also accessible from inside the same module only, even though the class **publicExample** can be accessed from anywhere because this class has **public** visibility by default.

Example

Let's check a simple example showing the usage of internal members:

```
package com.tutorialspoint.modifiers

open class A() {
   internal val i = 1

   internal fun doSomething(){
      println("Inside doSomething" )
```

```
        println("Value of i is $i" )
    }
}
class B : A() {
    fun printValue(){
        doSomething()
        println("Value of i is $i" )
    }
}

fun main(args: Array<String>) {
    val a = A()
    val b = B()

    a.doSomething()

    b.printValue()
}
```

When you run the above Kotlin program, it will generate the following output:

```
Inside doSomething
Value of i is 1
Inside doSomething
Value of i is 1
Value of i is 1
```

Quiz Time (Interview & Exams Preparation)

Q 1 - Which one is not a visibility modifier in Kotlin :

A - public

B - private

C - abstract

D - internal

Q 2 - Which one is correct about protected visibility modifier?

A - Protected members can not be accessed outside of the class

B - Protected members can be accessed by the subclasses

C - Kotlin does not allow to define a class as protected

D - Protected members can be accessed to its class or subclass only.